

THE OFFICE OF THE STATE CHIEF INFORMATION OFFICER
ENTERPRISE TECHNOLOGY STRATEGIES

North Carolina Statewide Technical Architecture

Domain White Paper
Data Architecture Technology Overview

STATEWIDE TECHNICAL ARCHITECTURE

Domain White Paper:

Data Architecture

Technology Overview

Initial Release Date:	August 1, 2003	Version:	1.0.0
Revision Approved Date:	Not Applicable		
Date of Last Review:	March 11, 2004	Version:	1.0.1
Date Retired:			
Architecture Interdependencies:			
Reviewer Notes: Reviewed and updated office title and copyright date. Added a hyperlink for the ETS email – March 11, 2004.			

© 2004 State of North Carolina
Office of the State Chief Information Officer
Enterprise Technology Strategies
PO Box 17209
Raleigh, North Carolina 27699-7209
Telephone (919) 981-5510
ets@ncmail.net

All rights reserved. No part of the material protected by this copyright notice may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying, recording or by any informational storage system without written permission from the copyright owner.

Mission Statement

The mission of Data Architecture is to establish and maintain an adaptable infrastructure designed to facilitate the access, definition, management, security, and integrity of data across the state.

Data and information are extremely valuable assets of the state. Data

Architecture establishes an infrastructure for providing access to high quality, consistent data wherever and whenever it is needed. This infrastructure is a prerequisite for fulfilling the requirement for data to be easily accessible and understandable by authorized end users and applications statewide.

Data and access to data are focal points for many areas of the Statewide Technical Architecture. Every application uses data in one way or another to operate. Data Architecture influences how data is stored and accessed, including online input and retrieval, outside application access, backup and recovery, and data warehouse access. (See Figure 1.) An established Data Architecture is the foundation for many other components of the statewide technical architecture, especially Application and Componentware Architectures.

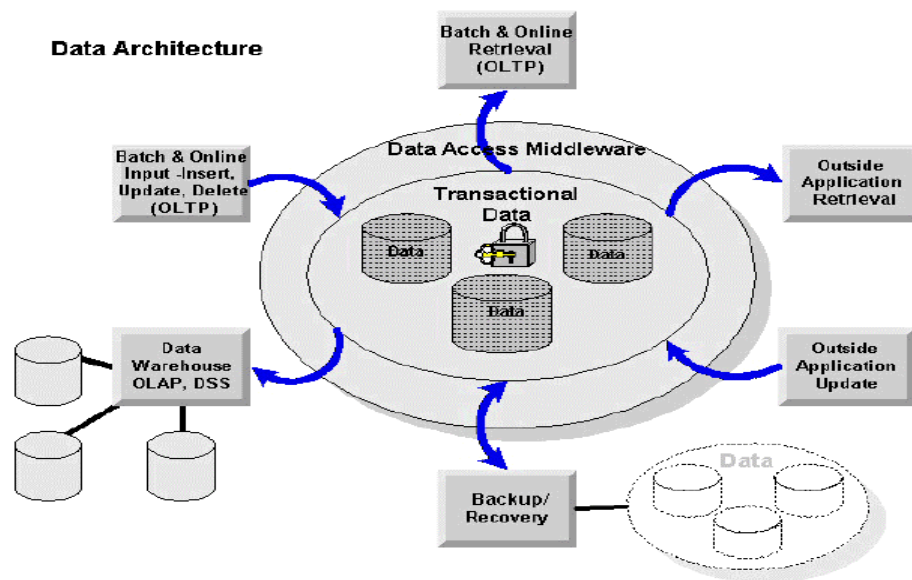


Figure 1. Data Architecture

As information technology has progressed throughout the state, the state's data has become distributed and defined differently by all the existing applications in all the different agencies. Data is stored on multiple platforms using multiple formats, application-specific designs, and a wide variety of semantics. Moreover, in the past, the data formats and semantics adhered to application-specific standards only, because no consistent agency or statewide standards were in place when the original applications systems and corresponding databases were developed.

When Data Architecture is not explicitly designed and actively managed, an unplanned data infrastructure emerges that may not meet existing and future requirements:

- Applications may directly access data managed by other mission critical transaction systems, thereby impacting the performance of those systems (and the accuracy of the data).
- There is little reuse and sharing of existing data and data management applications software. A significant number of resources are used to store and maintain the same data in many places.
- There is little, if any, central data administration applied to the data, so data is not consistently defined across the state.
- Sources for accurate up-to-date data (i.e., authoritative sources) are not identified, managed, cleansed, or accessible.
- It is difficult to identify how data flows through or is used by different applications, thereby making data management extremely difficult.

Therefore, it is important to ensure that the statewide Data Architecture is understood and used by project managers and application developers across the state.

To appreciate how data is stored and maintained, it is important to understand the types of applications that access data. There are two categories of application processing, online transaction processing (OLTP) and online analytical processing (OLAP).

When an application is used to perform mission-critical day-to-day operations, it is called an online transaction processing system. OLTP systems are typically used by many users simultaneously to perform data input, update, or retrieval. An OLTP system executes atomic business functions, performing finite units of work, typically in the form of one or several transactions at a time (e.g., a "renew vehicle registration" transaction or a "change citizen address" transaction).

When a system is used for analysis, planning, and management reporting through interactive access to a wide variety of information, it is called an online analytical processing system. An OLAP system usually references summary data to answer questions like "how much should be budgeted for office supplies next year?" or "how many traffic accidents occurred on Main Street last year?" Data for OLAP systems is typically extracted from OLTP applications and loaded or replicated to special databases that comprise a data warehouse. Data warehouses are specially designed to provide the type of information required for business analysis in an enterprise or agency.

Although common data infrastructure architecture applies to both OLTP and OLAP systems, there are special considerations for data stored and maintained in a data warehouse for OLAP access. For more information about data warehouse infrastructure and the processes involved in designing and maintaining a data warehouse, refer to the Data Warehouse Architecture chapter.

With business needs constantly changing, new systems under development, and existing systems being re-engineered, there is a growing need to make data available to multiple applications executing across agencies and across the state. To accomplish this goal, federated metadata should be used. Using federated metadata ensures that data is:

- Defined consistently across the state
- Re-useable and shareable
- Accurate and up-to-date
- Secure
- Centrally managed.

Federated metadata facilitates data sharing within a single agency, between multiple governmental organizations, and across the state. Federated metadata is covered in more detail in the Federated Metadata technical topic later in this chapter.

The Data Architecture consists of the following technical topics, including the recommended best practices, implementation guidelines, and standards, as they apply:

- Federated Metadata
- Data Modeling
- Database Management System (DBMS)
- Data Access Middleware
- Data Access Implementation
- Data Security

Federated Metadata Overview

- With the redesign of legacy systems to add new business functions, changing legislation, and the emergence of web applications, the Federated Metadata Repository has become a key enterprise tool for inter-and intra-agency collaboration. This topic provides an overview of the Federated Metadata Repository.

Data Modeling Overview

- How data is modeled and designed inside an application can significantly impact the way an application runs and how other applications can access that data. This topic covers a basic overview of data modeling for OLTP and OLAP applications.

Database Management System (DBMS) Overview

- Database Management System (DBMS) addresses the Data Architecture recommendations for projects selecting, designing, and implementing database management systems. In order to meet existing and future database needs, a relational database technology is recommended, particularly for online transactional business applications.
- An emerging technology in the database world is the object database technology. Object databases are data management products specifically designed for use with object-oriented programming languages. They provide DBMS capabilities to objects created by object programming languages, such as Smalltalk or C++. The DBMS topic provides an overview of object-oriented databases. This database technology will be documented further as the technology matures.

Data Access Middleware Overview

- Data access middleware addresses the Data Architecture recommendations for the implementation of data access middleware. Data access middleware is the communications layer between data access programs and databases.

Data Access Implementation Overview

- The Implementing Data Access topic is a key topic of this chapter. Data access is a fundamental component of every application. This topic discusses recommendations for implementing data access within an application and to outside applications.

Data Security Overview

- Data security is an important piece of the Data Architecture and the application security model. This topic provides an overview of data security and discusses the best practices for protecting data.

Federated Metadata

In the past, the state's applications were developed independently from each other. Application development was driven by federal or state legislative initiatives, program rules, or business needs. This form of application development is described as monolithic or stovepipe, where data is not reused and shared enterprise wide but collected and duplicated in numerous databases within a single agency and throughout the state. For example, names, addresses, and social security numbers are stored and maintained in every application that needs that particular data. It is difficult to determine which application's database stores the most current or correct information. Storing and maintaining redundant copies of the same data throughout the enterprise is time consuming and expensive.

In a typical monolithic or stovepipe environment, each application performs every step of the process needed to complete the entire business function. Applications do not share any logic or data across system or organizational boundaries. These databases are designed for access by single applications within a single agency, not for access by multiple applications in multiple agencies simultaneously. (See Figure 2.)

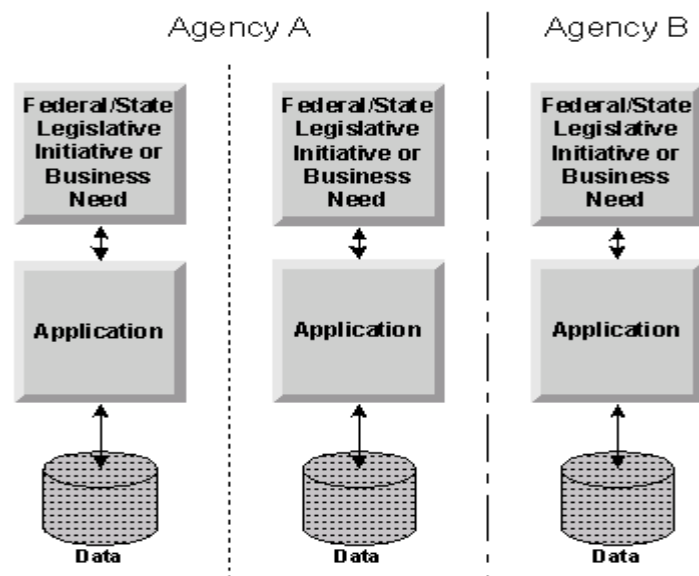


Figure 2. Monolithic or Stovepipe Application Development

The state has recognized the need for agency-wide and statewide data sharing and data comparison. One of the obstacles for sharing data across the enterprise is that the current data definitions implemented in existing databases have not been consistent. Data definitions are determined during the data modeling phase, which occurs early in the application development lifecycle. Over the years many different data modeling philosophies have been utilized. Therefore, many anomalies have been introduced into the way data has been defined and stored in databases. These data modeling anomalies exist not only across agency boundaries, but also within multiple applications in a single agency. Some examples of these anomalies are as follows:

- Identification. How a record is uniquely identified. An example of an identifier difference is where one database may store data about a North Carolina citizen using "Drivers License Number", while another may store data about the same citizen using "Social Security Number".
- Semantic. Different values associated with the same data element. An example of a semantic difference is where one database may refer to "Gender" as "Male" or "Female" and another may refer to "Gender" as "1" or "2".
- Synonym. Different field names for the same data element. An example of a synonym difference is where one database may refer to gender as "Sex" and another may refer to gender as "Gender".
- Homonym. The same field name for different data elements. An example of a homonym difference is where one database may refer to "Race" as it relates to a person and another may refer to "Race" as it relates to an election.

The state is now developing applications that cooperate and share data, both within a single agency and between agencies. In this model, common data elements are defined consistently even when they are stored in multiple databases and data can be shared between applications. This type of data is referred to as federated data.

When federated data is defined consistently, data is described the same way in each table where it is defined. (See Figure 3.) Definitions include traits such as name of the field, length, number format, data format, and the values it can have. When the data has the same format, it is much easier to exchange data across system and organizational boundaries.

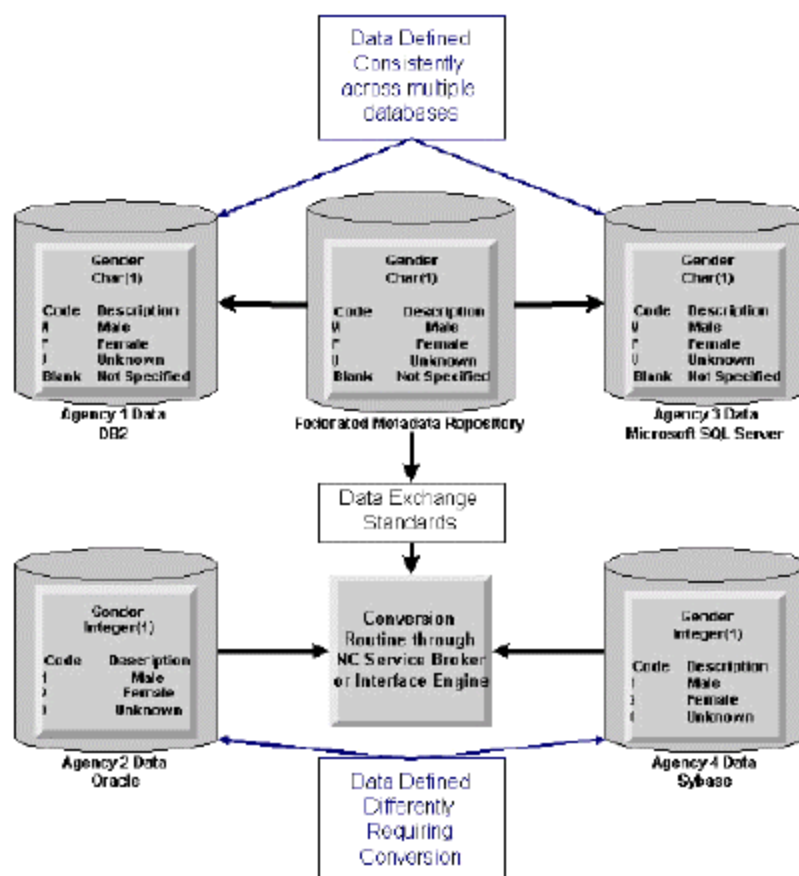



Figure 3. Using the Federated Metadata Repository (FMR) for Consistent Definitions and Exchange

The way to describe or define data is through metadata. Metadata is "information about data." Metadata is stored in a repository containing detailed descriptions about each data element. By using the formats described in the metadata repository, whether the data resides in a single location or in multiple databases across the state, the same data management principles apply. The state has an enterprise metadata repository called the Federated Metadata Repository (FMR). It is used to store agency and statewide metadata. Figure 4 shows an example of metadata for one statewide data element standard in the FMR.



Standards of the metadata
Proposed Elements
Data Values
Search
About our Metadata
Contact us
Home

- Find NC
- Research at
Metadata
- DDA CHIPS
- TEND Collaboration
- NC State metadata
Architecture


State of North Carolina

Federated Metadata Repository

Standard Data Element Details

Field	Values
Data Element/Attribute Name:	Social Security Number
Element Definition:	A 9-digit number assigned to an individual by the Social Security Administration
Business Format:	123-45-6789
Business Length:	11 positions
Business Type:	Number
Exchange Format:	123456789
Exchange Length:	9 Bytes
Exchange Type:	Character
Storage Format:	123456789
Storage Length:	9 Bytes
Storage Type:	Character or Variable Character

Powered by



Copyright ©2004-2005 The Open Technology, Inc. All rights reserved.

Figure 4. An FMR metadata example.

Federated metadata can be implemented even if data is physically located in disparate databases. Logically, if the data is modeled the same in each physical location, application interaction will be much less complicated. Federated metadata exchange standards simplify application integration for data sharing. When N-tier application design techniques are deployed, reusable services are implemented to access data that needs to be shared. When federated metadata exchange standards are deployed, data is used consistently in each application.

Shared data is accessed through reusable services called by each application that needs access to the data. Data can be shared through an interface, such as the North Carolina Service Broker (NCSB). To access the data, each application sends a request to the NCSB. The NCSB executes the service and returns a response to the application. This model protects the data and ensures that the data retrieved is accurate and consistent since it is being retrieved by proven application code.

The service used to access the data is typically written and owned by the application and business users who control the original data. The owners understand exactly how the data is stored and how to return accurate results. By writing a common service, the owners of the data ensure that it is accessed in the form intended and that inaccurate results are not returned to a user directly accessing the data who does not understand how the data is structured. For example, a service that returns drivers license information is one service available through NCSB. It was written by DOT and can be accessed by multiple authorized agencies requiring that information. (See Figure 5) The only access from outside applications to DOT

information is through this service, and no ad hoc access direct to the database is allowed. Thus, security and performance are not unknowingly jeopardized.

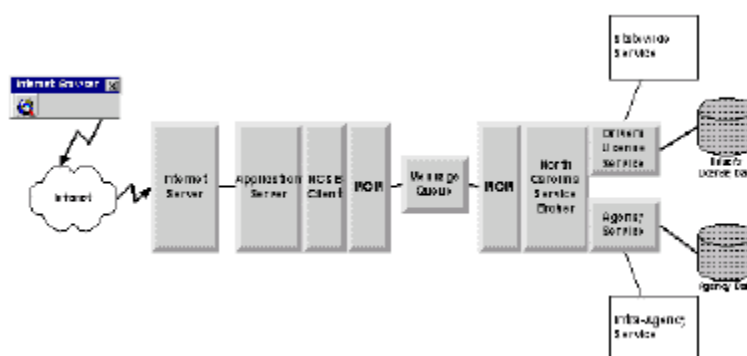


Figure 5. North Carolina Service Broker (NCSB) example

For more information about the NCSB, refer to the Application Communication Middleware Architecture chapter and <http://irm.state.nc.us/NCSB/>.

Ultimately, business and program users within governmental organizations across the state are responsible for the definition and the accuracy of data. Federated metadata is defined collaboratively by business users throughout the state. The Federated Metadata Repository effort to define federated metadata does not focus on developing a massive data infrastructure. The data models are being developed incrementally over time through ongoing projects. This effort provides shared resources for data accessed frequently by multiple applications.

The state's Metadata Element Review Team consists of key data architects from across the enterprise. The focus of the Metadata Element Review Team is to agree on the models and semantics of federated metadata to be shared throughout each agency and the state. In order to define federated metadata, the right people, who have both the knowledge and the authority to negotiate federated definitions of data, should be involved.

The Federated Metadata Repository stores information about proposed and standard data elements. It also identifies statewide and agency data elements. (See Figure 6.)

- Non-authoritative data elements are used by more than one agency, but no agency can be identified as the authoritative source. For example, "Social

"Security Number" is used in many applications and agencies across the state, but no state agency issues Social Security numbers.

- Authoritative data elements are used by more than one agency, but an agency can be identified as the authoritative source. For example, "Drivers License Number" is used by more than one agency, but DOT issues the drivers license and therefore is the authoritative source.
- The FMR can also store database information for each agency application. When database information is documented and available, information is more easily located and shared. By storing the appropriate database information in the FMR, agencies automatically comply with North Carolina Government Information Locator Service (NC GILS) and North Carolina Public Records Law (North Carolina G.S. § 132-6.1 (b)).

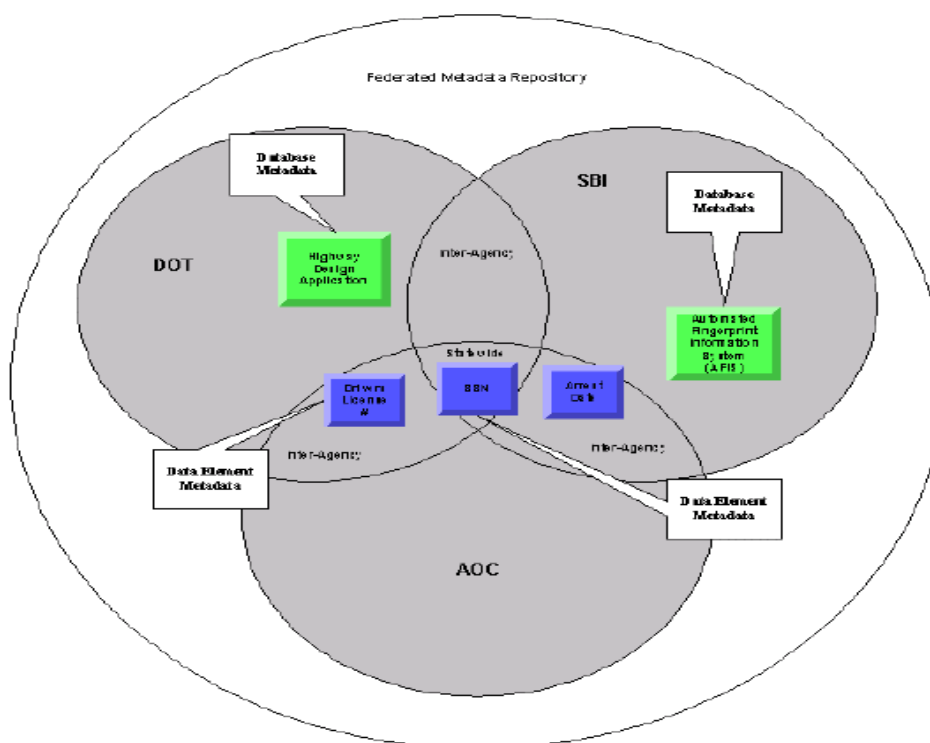


Figure 6. Sample Statewide and Inter-Agency Data Elements and Database Metadata Stored in FMR

An example of database information for an agency is shown below. (See Figure 7.)



Figure 7. APMS Database Information stored in FMR.

Federated Metadata Repository (FMR)

The Federated Metadata Repository provides a centralized location where all state agencies can store metadata. The IRM office maintains the FMR and it can be accessed through the Internet. The primary objectives of the FMR are as follows:

- Provide a central location to define/describe databases and data elements, enabling electronic compliance with the North Carolina Public Records Law and North Carolina Government Information Locator Service (NC GILS).
- Provide a resource for state employees and the general public to view how other applications are storing and using data.
- Provide a resource for state employees and the general public to view proposed and standard data element metadata.
- Facilitate data exchange using the North Carolina Service Broker (NCSB).

Data Modeling

The Data Modeling topic is designed to provide a basic overview of data modeling for OLTP and OLAP systems. It is not designed to be comprehensive, but it is designed to provide some best practices and implementation guidelines. There are reference books and tools for data modeling to provide further information.

Data modeling is the process of defining a data model for a project or application and is typically performed at the same time as the business model during the design phase of a project. An example of a data model is shown in figure 8.

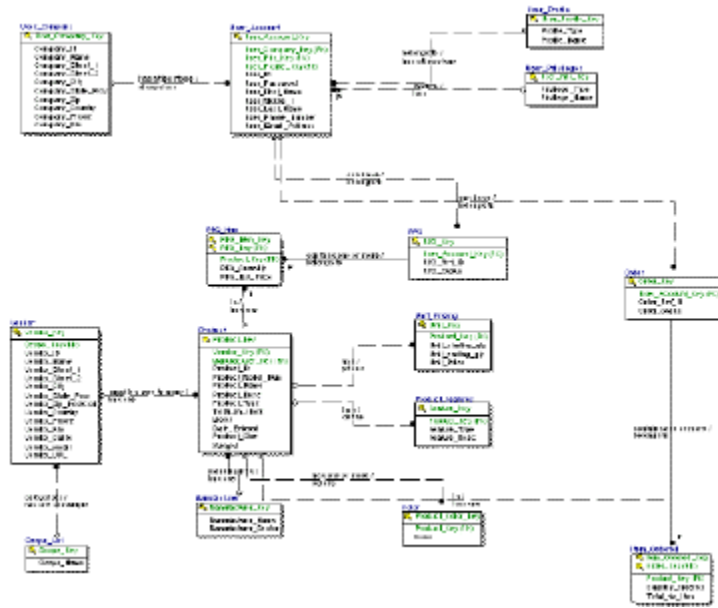


Figure 8. A sample data model of an order entry system.

The roles and responsibilities related to the business model are as follows:

Function	Personnel		
	Analyst	Database Analyst (DBA)	Programmer
Analyze/Document Process	Primary	Provide Support	
Create Business/Data Access Rules	Primary	Provide Support	
Create Logical Data Model	Provide Support	Primary	
Create Physical Data Model	Provide Support	Primary	
Build and Support the		Primary	

Database			
Create stored procedures only if necessary (see Data Access Implementation Topic)	Provide Support	Primary	
Program Business Rules/Data Access Rules in the middle tier(s)	Provide Support		Primary

Table 1. Roles and Responsibilities for Data Modeling

A data model is a framework used to collect and analyze data requirements and consists of entities, attributes, relationships, and cardinalities:

- An entity is a person, place, thing, or concept and becomes a table in the database. A citizen is an example of an entity. A row in a table is an instance of an entity.
- An attribute is a characteristic that provides further information about the entity like who, what, when and where. It becomes a column in the database. A key attribute, or primary key, uniquely identifies an entity; so the values are distinct for each individual entity. Examples of primary keys are Drivers License Number and Serial Number. A key attribute in a database cannot be null. A non-key attribute further describes entity but does not uniquely identify the entity (e.g., female, sedan, blue). A non-key attribute can be null in a database.
- A relationship is how one entity is related to another. A relationship is shown through verbs or verb phrases because a relationship represents an action. Examples of relationships are: "owns/owned by", "insured by/insures". When a relationship is established between two entities, there is a parent entity and a child entity.
- Cardinality is how many occurrences of an entity to expect in a relationship. There are two types of cardinality, conditional and unconditional.

Conditional cardinality is where a relationship may or may not exist. The child entity is not required. Conditional relationships are defined as follows:

- Zero, one or many (e.g., a citizen owns zero, one or many vehicles).
- Zero or one (e.g., a student can register no vehicles or only one vehicle to park on campus).

Unconditional (mandatory) cardinality is where a relationship always exists. In this case, the child entity is required. Unconditional relationships are defined as follows:

- One-to-one (e.g., a county has one and only one county seat).
- One-to-many (e.g., a school has one or more teachers).
- Many-to-many (e.g., a classroom can be assigned to many students and students can be assigned to many classrooms). Many-to-many relationships can be simplified through an associative entity (e.g., a classroom-student entity).
- One-to-Specific-Number (e.g., one student has exactly 6 classes).

An identifying relationship connects two dependent entities. For example, a vehicle would not be in the database without an owner. It is usually indicated in a data model by a solid line. In an identifying relationship, the foreign key becomes part of the primary key.

A non-identifying relationship relates two independent entities (e.g., an insurance company can exist in the database without a vehicle associated with it). It is usually indicated in a data model by a dotted line. The foreign key becomes a non-key attribute and is not part of the primary key. A recursive relationship is where the entity is both the parent and the child and the relationship is non-identifying. An example of a recursive relationship is where a product can be constructed of multiple products stored in the same table.

Logical vs. Physical Data Modeling

Logical data modeling is a business perspective of a project's data that is independent of how it will be stored in the database. It describes the business data requirements in a format where business users can understand it.

Physical data modeling is a data model that directly represents the logical data as it will be stored in the DBMS. The physical model is derived from the logical model. The methods for migrating from a logical to a physical model are as follows:

- The entities become database tables and the attributes become database columns.
- Data types and field lengths are defined for each attribute.
- Indexes are created and system-enforced referential integrity rules are defined.
- Data definition language (DDL) is created to define the database and associated tables.
- The tables are populated with domain values, which are the pre-defined values for particular attributes (e.g., colors or models).

Different types of data modeling techniques are used for online transaction processing (OLTP) and online analytical processing (OLAP). OLTP typically uses Entity-Relation (ER) Modeling, where OLAP typically uses a combination of dimensional modeling and ER modeling. Most OLAP products and many other front-end user tools require a star or snowflake schema. For more information about OLTP and OLAP, refer to the introduction of this chapter and the Data Warehouse Architecture chapter.

Entity-Relation (ER) Modeling

Entity-relation (ER) Modeling produces a data model of entities, relationships, and attributes. A primary goal of ER modeling is to reduce or remove data redundancy, where the same data is stored multiple times in a database and is therefore difficult to maintain data integrity. It is used for OLTP because it simplifies transaction complexity. For OLAP processing, ER modeling is used for the transaction capture and data administration phases of constructing a data warehouse. ER modeling should be avoided for end user delivery of OLAP applications. In this case, dimensional modeling should be used.

Entity-relation (ER) normalization is a method that organizes attributes to form stable, flexible, and adaptive entities. ER normalization has many levels, but most data models are normalized to the third level and then denormalized where needed for performance reasons. Each level is referred to as a normal form and the first three levels are first normal form, second normal form, and third normal form. In an ER model, transformation from the logical model to the physical model is straightforward.

A simple example of normalizing data is organizing basic data related to a citizen with county, vehicle, and vehicle insurance information. Figure 9 shows the data fields and instance table that will be normalized.

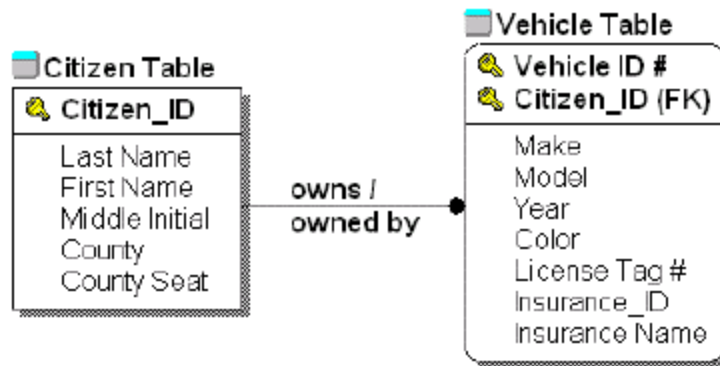
Citizen Table

Citizen ID	Last Name	First Name	Middle Initial	County	County Seat	Vehicle ID #	Make	Model	Year	Color	License Tag #	Insurance ID	Insurance Name
------------	-----------	------------	----------------	--------	-------------	--------------	------	-------	------	-------	---------------	--------------	----------------

Citizen ID	Last Name	First Name	Middle Initial	County	County Seat	Vehicle ID Number	Make	Model	Year	Color	License Tag	Insurance ID	Insurance Name
1	Baske	Pat	R	Mecklenburg	Charlotte	ABC123456	Ford	Explorer	1994	Blue	ABC1234	5	Hamer
2	Dandson	Carol	H	Durham	Durham	DEF456789	Chrysler	Saturn	1986	Red	DEF456	4	State Farm
3	Davis	Chris	W	Orange	Chapel Hill	XYZ987654	Toyota	Camry	1997	Green	GHI567	3	GEICO
3	Davis	Chris	W	Orange	Chapel Hill	PQR987654	Chrysler	Jeep	1990	Black	JKL4567	2	Allstate
4	Do	John	J	Guilford	Greensboro	GHI234567	Bmw	300i	1996	White	MNO5678	4	State Farm
5	Johnson	Fred	A	Forsyth	Winston-Salem	JKL345678	Chevrolet	Suburban	1972	Grey	PQRS789	3	GEICO
6	James	James	E	Columbia	Winterville	MNO456789	Acura	Integra	1991	Tan	STU9890	5	Hamer
7	Parkes	Ann	E	Wake	Raleigh	STU567890	Honda	Accord	2000	Silver	VW98701	2	Allstate
7	Parker	Ann	E	Wake	Raleigh	VW98701	Chevrolet	Lumina	1990	White	YZAC012	2	Allstate
8	Smith	John	T	Forsyth	Louisburg	TUV989012	Ford	Mustang	1990	Black	BCD0123	4	State Farm

Figure 9. Normalization example - Denormalized data model.

The first step is to take the model to the first normal form (1NF). In the first normal form, each attribute or column occurs only once for each instance of the primary key. Since a citizen can own multiple vehicles, the vehicle data moves to a new entity called Vehicle. (See Figure 10.) This way, the data for each citizen is stored only once and multiple vehicles owned by a citizen can exist without citizen data repeated. Characteristics of first normal form are as follows:

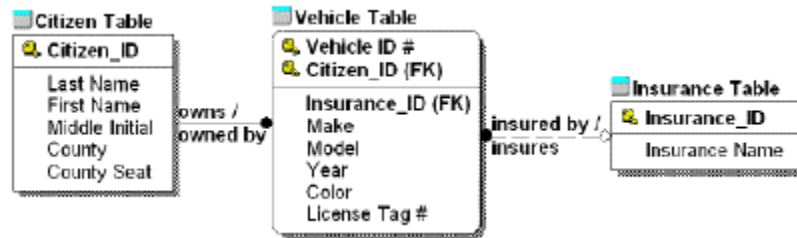


Citizen Table								
Citizen ID	Last Name	First Name	Middle Initial	County	County Seat			
1	Baker	Pat	R	Mecklenburg	Charlotte			
2	Davidson	Carl	H	Durham	Durham			
3	Davis	Chris	W	Orange	Chapel Hill			
4	Doe	Jane	J	Guilford	Greensboro			
5	Johnson	Fred	A	Forsyth	Winston-Salem			
6	Jones	Jeanne	M	Columbus	Whiteville			
7	Parker	Ann	E	Wake	Raleigh			
8	Smith	John	T	Franklin	Louisburg			

Vehicle Table								
Vehicle ID Number	Citizen ID (FK)	Make	Model	Year	Color	License Tag	Insurance Company ID	Insurance Company
ABC123456	1	Ford	Taurus	1994	Blue	ABC1234	5	Huon
DEF456789	2	Chrysler	Sebring	1995	Red	DEF2345	4	State Farm
XYZ987654	3	Toyota	Camry	1997	Green	GHI3456	3	GEICO
PQR87654	3	Chrysler	Jasp	1992	Black	JKL4567	2	Allstate
GHI234567	4	BMW	320si	1998	White	MNO5678	4	State Farm
JKL345678	5	Chevrolet	Suburban	1972	Grey	PQR6789	3	GEICO
MNO456789	6	Acura	Integra	1991	Tan	STU7890	5	Huon
STU567890	7	Honda	Accord	2000	Silver	VWX8901	2	Allstate
VWX678901	7	Chevrolet	Lumina	1993	White	YZA9012	2	Allstate
TUV789012	8	Ford	Mustang	1999	Black	BCD0123	4	State Farm

Figure 10. Normalization example - First Normal Form (1NF).

The next step is to take the model to second normal form (2NF). Second normal form starts with the first normal form model, then any partial dependencies are removed so all the non-key attributes are fully dependent on the composite primary key. In the example, the insurance company is not dependent on the composite primary key, so it is put into a separate entity. (See Figure 11.) By definition, a model is already in second normal form if it only has a single attribute primary key.



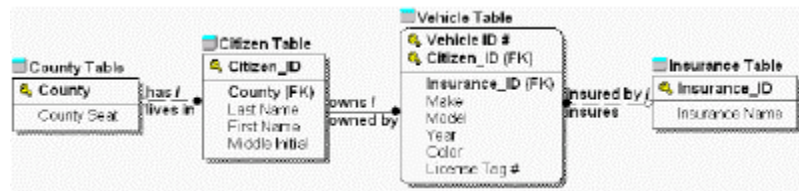
Citizen Table					
Citizen ID	Last Name	First Name	Middle Initial	County	County Seat
1	Baker	Pat	R	Mecklenburg	Charlotte
2	Davidson	Carol	H	Durham	Durham
3	Davis	Chris	W	Orange	Chapel Hill
4	Doe	Jane	J	Guilford	Greensboro
5	Johnson	Frank	A	Forsyth	Winston-Salem
6	Jones	James	E	Columbus	Wilmington
7	Farther	Ann	T	Wake	Raleigh
8	Smith	John	T	Franklin	Louisburg

Vehicle Table							
Vehicle ID Number	Citizen ID (FK)	Make	Model	Year	Color	License Tag	Insurance Company ID (FK)
ABC123456	1	Ford	Taurus	1994	Blue	ABC1234	5
DEF456789	2	Chrysler	Sebring	1995	Red	DEF2345	4
XYZ987654	3	Toyota	Pammy	1997	Green	GHI3456	3
PQR876543	3	Chrysler	Jeep	1992	Black	JKL4567	2
GHI345678	4	BMW	320i	1998	White	MNO5678	4
JKL345678	5	Chevrolet	Suburban	1972	Gray	PQR6789	3
MNO456789	6	Acura	Integra	1991	Tan	STU7890	5
STU567890	7	Honda	Accord	2000	Silver	VWX8901	2
VWX678901	7	Chevrolet	Lumina	1993	White	YZA9012	2
TUV789012	8	Ford	Mustang	1999	Black	BCD0123	4

Insurance Table	
Insurance Company ID	Insurance Company
2	Allstate
3	GEICO
4	State Farm
5	Hursh

Figure 11. Normalization example - Second Normal Form (2NF).

The final step is to take the model to third normal form (3NF). Third normal form starts with the second normal form model and then modifies it so that no attributes depend on any other non-key attributes. As well, there are no attributes calculated from other attributes. In third normal form, all attributes are dependent on the primary key and only the primary key. In the example, the county seat is dependent on the county, so a separate county entity is added. (See Figure 12.)



Citizen Table								
Citizen ID	Last Name	First Name	Middle Initial	County (FK)				
1	Baker	Pat	R	Northburg				
2	Davidson	Carol	H	Durham				
3	Davis	Chris	W	Orange				
4	Doe	Jane	J	Guilford				
5	Johnson	Fred	A	Forsyth				
6	Jones	James	E	Columbus				
7	Patton	Aria	B	Wake				
8	Smith	John	T	Franklin				

County Table								
County	County Seat							
Columbus	Whiteville							
Durham	Durham							
Forsyth	Winston-Salem							
Franklin	Louisburg							
Guilford	Greensboro							
Hicklenburg	Chatham							
Orange	Chapel Hill							
Wake	Raleigh							

Vehicle Table								
Vehicle ID Number	Citizen ID (FK)	Make	Model	Year	Color	License Tag	Insurance Company ID (FK)	
ABC123456	1	Ford	Taurus	1994	Blue	ABC1234	5	
DEF456789	2	Chrysler	Stratus	1995	Red	DEF4567	4	
XYZ987654	3	Toyota	Camry	1996	Green	GHI9876	3	
PQR567890	3	Chrysler	Jeep	1992	Black	JKL4567	2	
GHI234567	4	BMW	320si	1998	White	MNO6789	4	
JKL345678	5	Chrysler	Suburban	1992	Grey	PQR5678	3	
MNO67890	6	Acura	Integra	2001	Tan	STU9012	5	
STU90123	7	Honda	Accord	2000	Silver	VWX8901	2	
VWX89012	7	Chrysler	Lancia	2002	White	YZA9012	2	
TUV789012	8	Ford	Mustang	1999	Black	RST0123	4	

Insurance Table								
Company ID	Company							
2	Allstate							
3	GEICO							
4	State Farm							
5	Hart							

Figure 12. Normalization example - Third Normal Form (3NF).

Once a model has reached third normal form, any variation from that form is called denormalization. Queries against an ER model typically require the joining of two or more tables together. The more joins required for a query, the more chance of performance degradation. Denormalization is sometimes necessary in a data model to increase the performance of data access.

Dimensional Modeling

Dimensional modeling is a more abstract data modeling technique than entity relation modeling, using facts, dimensions, hierarchies, and attributes like how much, how many, and how often. Dimensional modeling is used to design databases that support ad hoc queries in a data warehouse environment.

In a dimensional model, data is denormalized and stored redundantly. Even though duplicate data takes up more space, it reduces the number of joins required to process a request, so it reduces the processing time required. In addition, indexes can be pre-built to support anticipated queries. Summary and calculated data can be provided as well, reducing the amount of time to calculate and summarize data. If there are a large number of rows, horizontal partitioning can be used to split a large table into two different tables at the row level. If the data has a large number of columns, vertical partitioning can be used to split a table into two different tables at the column level.

A dimensional model is composed of the following:

- **Fact table.** A fact table is a set of facts grouped together, defined with a multi-part primary key and usually consisting of multiple foreign keys. Most fact tables contain one or more numerical or additive measures, or "facts," that occur for the combination of keys that define each record. Most useful facts in a fact table are numeric and additive. Data warehouse applications/users rarely retrieve a single fact table record. Typically, they retrieve hundreds, thousands, or even millions of records at a time, and the ability to add up the data (additivity) is essential.
- **Dimension table.** Dimension tables most often contain summary or descriptive information that relates to a fact. Dimension tables are descriptive attributes about the facts can be grouped into one or more common structures
- **Schema.** A schema is the diagram of how the entities relate to each other. Common schemas are star and snowflake. A star schema is an arrangement of entities with a central fact table and related dimensions, and is usually used to build data marts. A snowflake schema is an arrangement of entities similar to the star schema but has additional secondary dimensions, and is typically used in data mining functions.

For a dimensional model example, a fictional data mart will be created for student test scores. In this example, time, school, student, grade level, and curriculum will be tracked. Figure 4-13 shows the sample star schema that could be used.

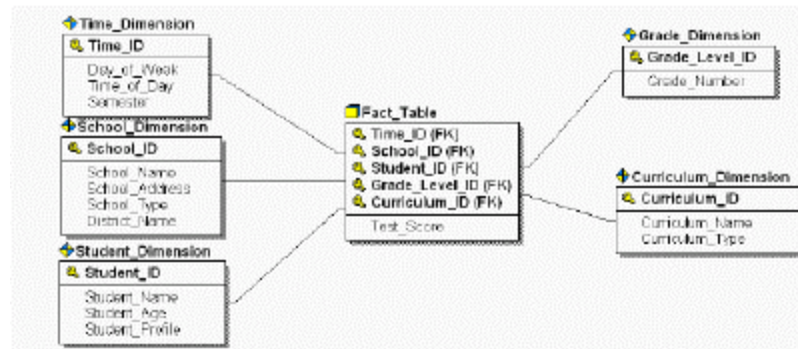


Figure 13. A sample star schema dimensional model.

Snowflaking adds hierarchical dimensions to the star schema by placing additional text attributes or columns into secondary dimensions. For example, snowflaking would add a District dimension table that extends the School dimension. Using the sample star schema, by adding a dimension, a snowflake schema begins to emerge. (See Figure 14.)

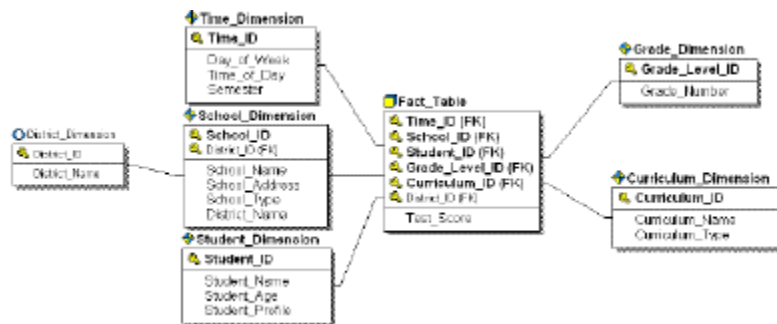


Figure 14. The emergence of a snowflake schema.

An extended snowflake schema makes use of multiple fact tables. As data models become more comprehensive, more fact tables will be defined. Large queries can use two or more fact tables linked by multiple dimension tables. The join of multiple snowflake schemas is referred to as a "snow storm".

Database Management System (DBMS)

Data is organized and managed through a database management system (DBMS). The database organization can be relational or non-relational. A DBMS manages data storage, structure, access, and security. Fields can be indexed to improve the performance of queries against the data, comparable to how a dictionary has tabs

for the letters of the alphabet so that it is easier to look up a word, or a reference book has an index to quickly find information needed.

A relational database management system (RDBMS) is a collection of data organized into related tables so relationships between and among data can be established. For example, a vehicle database can contain two tables, one for customer information and one for vehicle information. An "owns" relationship is then established between the two tables (e.g., "John Smith" owns a vehicle with the identification number "AB1CD234EF567890". (See Figure 15.)

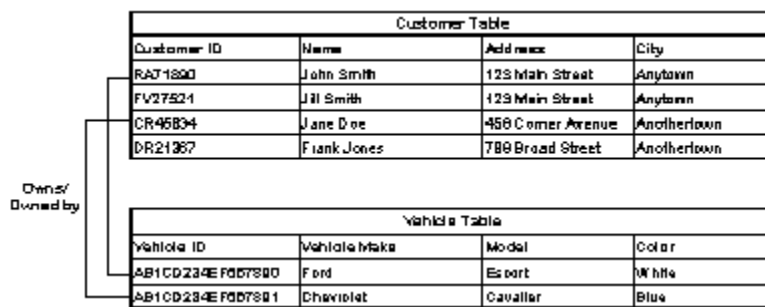


Figure 15. Relational Tables

Relational databases are specifically designed to store normalized data. Normalized data is organized so that unique data is stored only one time, instead of multiple times for each table (e.g., a non-normalized data). A relationship is established between the unique data and its related information. (See Figure 16.)

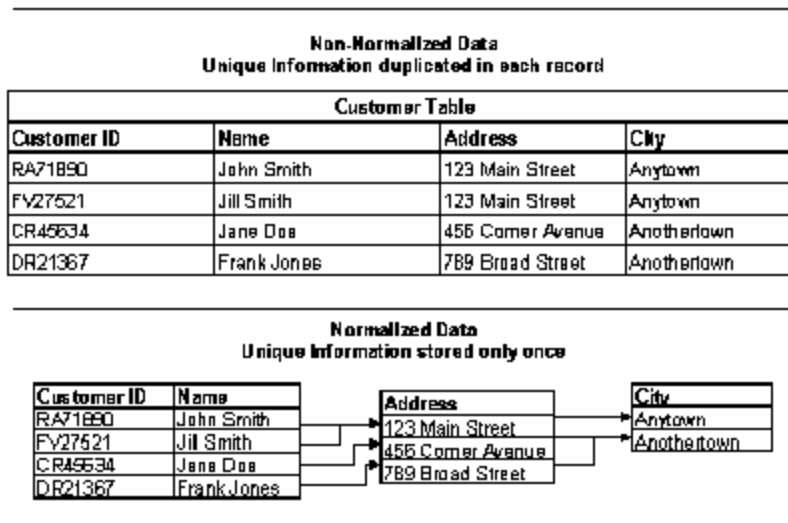


Figure 16. Database Organization - Non-normalized vs. Normalized Data

For more information about normalizing data, refer to the Data Modeling topic in this chapter.

When new databases are implemented in the state, relational database technology should be used, because of the efficiency, flexibility, and compatibility associated with relational technology. Relational databases should also be used when existing (legacy) data stores are re-engineered. Non-relational technology, particularly flat files, should only be used for unstructured data, textual data, and temporary work storage. The relational technology is needed to support the other components of the statewide technical architecture, including Application, Componentware, and Middleware Architectures.

The Application Architecture recommends an N-tier design for applications. In an N-tier design, data access services are implemented in a separate tier from business rules and user interfaces. To maintain consistent standards for how data access services access RDBMSs, and to ensure the same standards for end user access to data, all data access to the new relational data stores must be through ANSI-standard SQL, not proprietary SQL extensions. (See Figure 17.)

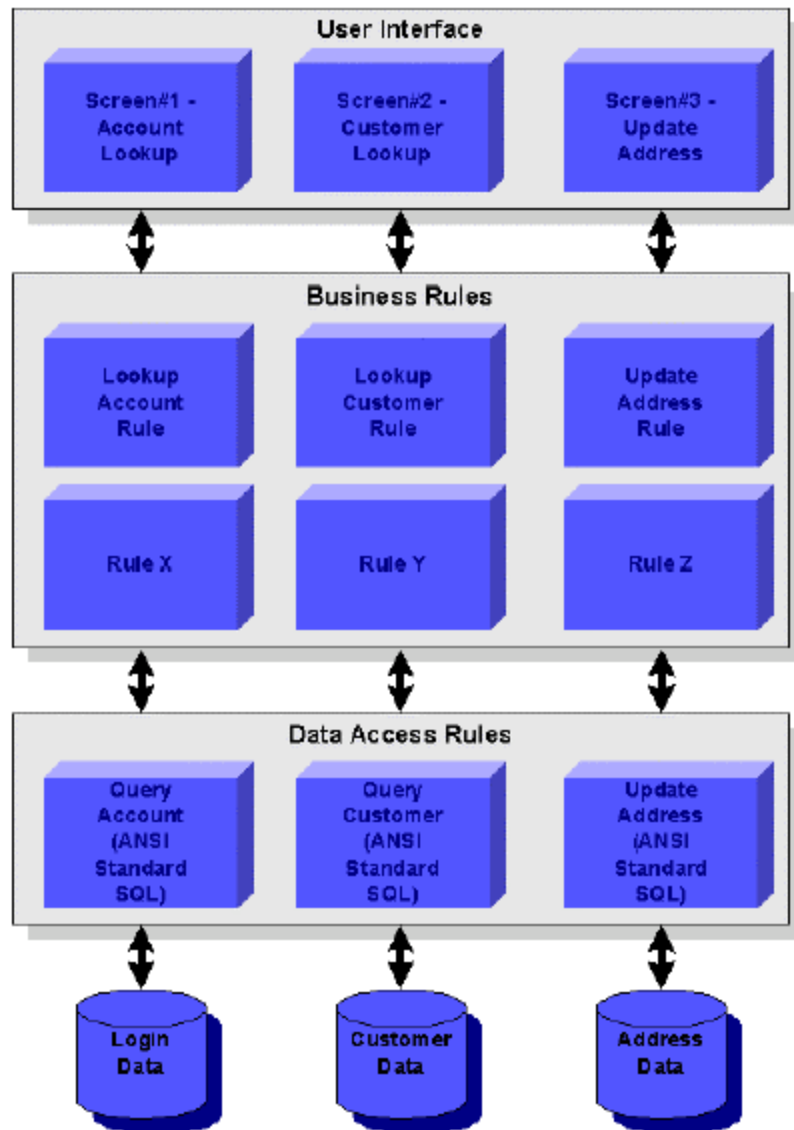


Figure 17. ANSI-standard SQL Access.

Replication

Replication is used to keep distributed data up to date with a central source database. Replication uses a database identified as a central source and reproduces the data to distributed target databases. Replication services are available from most relational database vendors. Applications and end users must access replicated data in a read-only mode. If updates are allowed on replicated data, data can quickly become corrupted and unsynchronized. Updates must be directed to the database

access tier in charge of updating the authoritative source, rather than a replicated database.

Object Database Management System (ODBMS)

In conjunction with object-oriented programming languages like C++, Java, and Smalltalk, another database structure is emerging called ODBMS. This type of database technology can be used to support data with very complex and nontraditional relationships. Since the ODBMS market is still considered a niche market, a waiver must be obtained in order to use this technology.

Although most object-oriented applications use relational databases to store and manage data, some use object databases to store data in the form of objects. Object databases store data in the same model and format as the object model, providing a seamless integration of objects and data. Objects in an ODBMS imitate the attributes and characteristics of the objects in real life, so it is easier to support more complex relationships and data types. When using a relational database with an object-oriented application, an object-relational mapping is required that maps the objects to the relational model.

A drawback of using ODBMS is that since the object defines each data type, data is stored in a proprietary format and the data cannot be understood without the object code that describes it. If new relationships between objects develop, adapting the object code may be difficult.

ODBMS is used in cases where the data model is very complex and unable to be mapped in a traditional relational format. The object database understands object-oriented concepts such as class hierarchy, inheritance, encapsulation, and polymorphism. ODBMS is typically used for complex systems such as manufacturing, hospital record processing, financial portfolio risk analysis, and telecommunications applications. Since this technology is still emerging, RDBMS technology is the standard.

Although some of the relational database vendors have extended their relational database technology to add support for storing objects it is still not possible to anticipate the impact this type of technology will have in the DBMS market space.

Data Access Middleware

Data access middleware is the communication layer between data access rules and the data itself. Data access middleware is designed to enable communication between a data access tier and a database, as opposed to application communication middleware, which enables communication between the programming tiers of an N-tier application. (See Figure 18.)

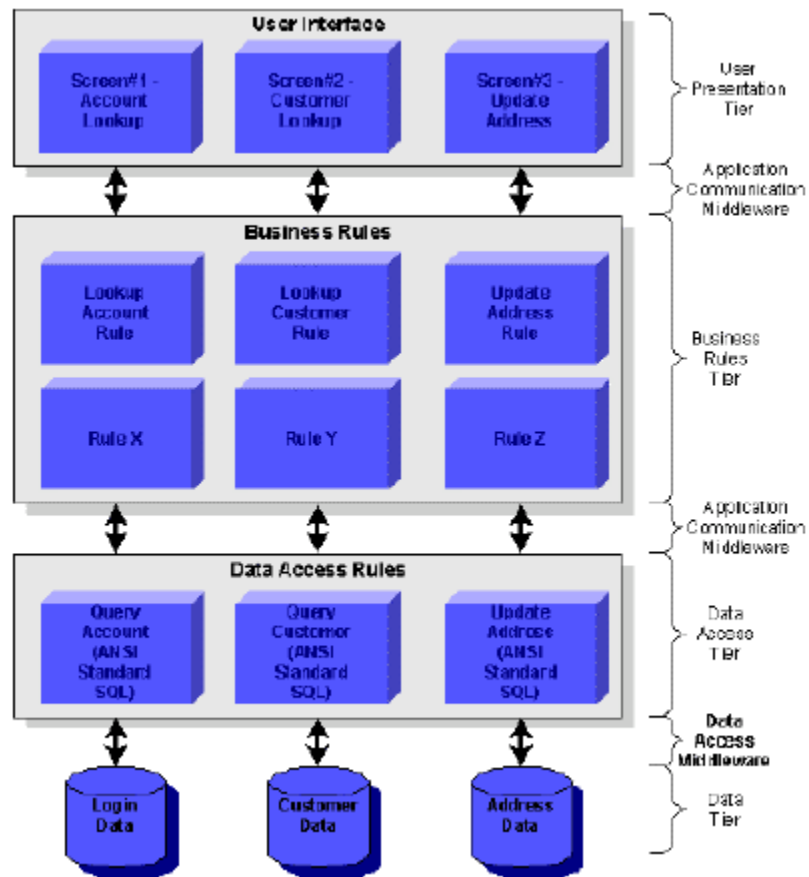


Figure 18. N-tier application environment

Database-Specific Middleware Drivers

Structured query language (SQL) is a query language used to query and retrieve data from relational databases. The industry standard for SQL is ANSI Standard SQL. SQL drivers are implemented by each RDBMS vendor to enable database access to its proprietary database (e.g., SQL*Net is Oracle's SQL driver, Open Client is Sybase's SQL driver). Vendors may add extensions to the SQL language for their proprietary databases.

Open Database Connectivity (ODBC) Drivers

Open database connectivity (ODBC) drivers are the middleware used to connect database access rules to relational databases through the use of a generic application program interface (API). ODBC drivers are vendor-provided and allow databases to be connected and used by a generic interface. The ODBC drivers enable access to data and provide insulation between a program and the specific RDBMS language used by each database. Database access tools and programs do not have to be customized for each database, because an ODBC configuration file maintains the database connections. ODBC is language-independent; so many different programming languages can use it.

ODBC can be implemented as a client-based solution or an application server-based solution. The application server-based solution is recommended by the Data Architecture. (See Figure 19.)

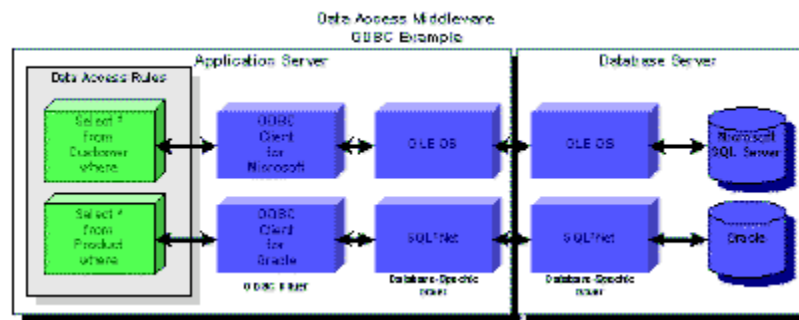


Figure 19. A sample application server-based SQL solution using ODBC

Note: For application access to data, refer to the Application Architecture chapter. For more information about application communication middleware, refer to the Application Communication Middleware Architecture chapter.

OLE DB

ODBC is part of the COM+ Microsoft specification called Object Linking Embedding Database (OLE DB). OLE DB provides specifications for an object-oriented API to access relational and object-oriented databases. Microsoft introduced OLE DB as a universal technology to query and update data in all databases of an enterprise, regardless of where and how the data is stored. Both relational and object-oriented databases are supported by the OLE DB specifications.

Data Access Implementation

Since data is at the core of most applications, data access is a vital component of the Data, Application, and Componentware Architectures. Depending on the application, data can be stored and accessed in numerous databases in multiple locations. When data is centralized or when data is distributed across an organization, data access must be carefully implemented with usability, accessibility, cost, performance, adaptability, and security in mind. This topic provides an overview of the different types of data access and discusses practices and guidelines to use when implementing data access including:

- Overall data access methods
- Specific data access methods
- Data integrity
- Data access design considerations.

Overall Data Access Methods

In a typical n-tier environment, the programs that perform data access are separate from the programs that perform business rules. Depending on the location, format, and owner of the data to be accessed, any of the following methods may be appropriate (See Figure 20.):

- 1 - Data access through data access rules. Programs performing data access rules can be executed as part of an n-tier application.
- 2 - Remote data access through the North Carolina Service Broker (NCSB). The NCSB can be used for inter-agency and intra-agency data sharing. In this case, the data is not accessed directly by the requesting application, it is accessed through a shared service.
- 3 - Remote data access through a shared agency service. An agency may have an existing service that can be called for data access. As is the case with NCSB, the data is not accessed directly by the requesting application, it is accessed through a shared service.
- 4 - Data access through a stored procedure. A stored procedure in a database can be called from an application or embedded SQL to perform data access. Stored procedures are not recommended as they create logic that is specific to a particular vendor and do not support vendor neutrality.

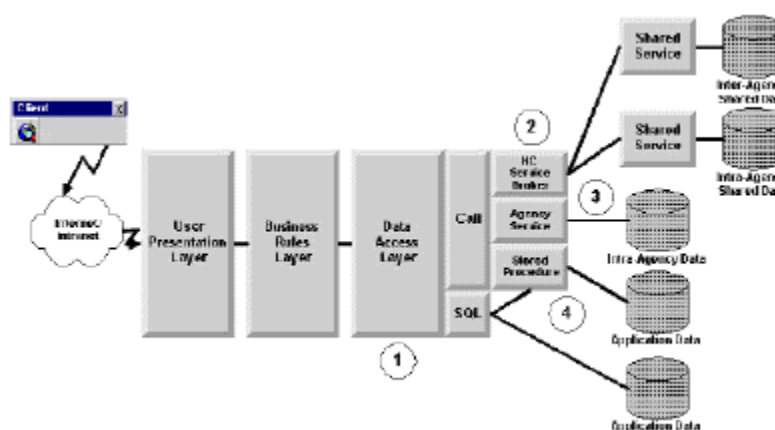


Figure 20. Application Architecture and Examples of Data Access.

Specific Data Access Methods

Data access can be implemented through various methods, including:

- **Embedded Structured Query Language (SQL).** Embedded SQL (ESQL) is a program that contains a combination of SQL statements and other application query logic to process a request. Embedded SQL can be dynamic or static. When embedded SQL is used, a multi-step compile process is required, where the SQL is pre-compiled before the host language program is compiled.
- **Static Structured Query Language (SQL).** Static SQL is embedded SQL written into the source code of an application that does not change during application execution, (i.e., it remains static). Static SQL is used when the data access requirements are known during application design and compile time. Static requests are typically more efficient than dynamic SQL.
- **Dynamic Structured Query Language (SQL).** Dynamic SQL is embedded SQL that is built at application run-time. It is not written in the application source code, the SQL statements are composed when the application executes based on client input. Dynamic SQL is used when data access requirements are not known at design and compile time. Dynamic SQL is typically used in OLAP style applications, such as spreadsheets and ad hoc access. Dynamic SQL can generate a significant impact on performance and security.
- **Call-Level Interface (CLI).** A call-level interface (CLI) is a library of DBMS functions that can be called by an application. CLI is an alternative to embedded SQL and is typically used in n-tier application design, where the business rules and data access rules are separate from the database server. CLI can generate a significant impact on performance and security.

- Stored procedures and triggers. Most database vendors provide the capability to store logic and data access rules in a stored procedure or trigger. Stored procedures are programs embedded in a table that can be called by an application. Triggers are programs embedded in a table that are automatically invoked by updates to another table. Stored procedures must be avoided, because the use of stored procedures creates database vendor lock-in. Since the stored procedures are stored in the database and are database-specific, they are not easily migrated if a new database platform is selected.

Data Integrity

Data integrity can be maintained by the DBMS or maintained by the application. Therefore, data integrity can be implemented inside the database and through data access rules.

- Entity integrity keeps duplicate records from being inserted in a table and is enforced through primary keys and normalized database designs.
- Domain integrity means that only valid values are entered into a field. It is enforced through foreign key constraints, column-level rules, or lookup tables.
- Referential integrity is making sure that no foreign keys point to records that do not exist. It is enforced using foreign key constraints.
- Business rules integrity is applying additional rules to data as it specifically relates to the business. These rules may cross column, row, and even table or database boundaries.

Design Considerations

Unless the database technology is outdated or there are performance problems with the selected database, the database platform is usually a stable environment, changing infrequently. However, data models are much more volatile. Data models change almost as frequently as the business needs change. Therefore, it is important when implementing data access to ensure the infrastructure and data access logic is easily adapted to changes in data models and business needs.

In order to properly separate the data access tier from the business rules tier, consideration must be given as to how much the application would be impacted if the data model or the database platform changes. If there is a high impact, the data access rules are more tightly coupled with the business rules; therefore, the data access design is less adaptable. If there is a low impact, business rules are loosely coupled with data access rules; therefore, the data access design is more adaptable. It is important to design an application so that there is a low impact if the data model or database changes.

Outside applications should have no direct access to data. When data sharing is required, there are two basic options:

- North Carolina Service Broker (NCSB). NCSB is a statewide facility designed to share common services and share data across the state. The NCSB can also be used inside of an application or within an agency. It is extremely useful in custom-designed applications where the application can be designed to use the messaging and service capabilities of the NCSB.
- The state's interface engine can be used in instances where the source code is unable to be modified or the data layouts are unavailable. It is an unobtrusive interface to accomplish data sharing.

For more information about NCSB and the interface engine, refer to the Componentware and Integration Architecture chapters.

When designing an application, particularly one that will be accessed through an Intranet or Internet, there are several design considerations:

- Performance. What are the performance needs of the end user?
- Availability. When does the application need to be up and running? How long can database be unavailable or unresponsive?
- Scheduling/Batch windows. What intervals are scheduled to execute batch processes, such as batch processing? What happens if the batch process is unable to finish in the allotted time?
- Backup. If an application requires 24X7 availability, a special database design needs to be considered; allowing backup of the database while it is still running.
- Replication. If replication is involved, how will the replication occur, (e.g., near real time, hourly, every night, etc.)?
- Language. Evaluate the data requirements and the needs of the end users to see if multi-lingual options need to be provided and how.
- Data Security. What are the data security requirements for the data? For more information about data security, refer to the Data Security topic.
- Database Recovery. How to achieve business continuity if something happens to the database.
- Disaster Recovery. How to achieve business continuity if something happens to the hardware or facilities.

Data Security

The state's data is a very valuable resource, and establishing a secure data environment is a key component of the Statewide Technical Architecture, particularly since more and more applications use the Internet to access data. It is critical that the state's data be protected against any unauthorized access.

Data security is designed to protect data against the following threats:

- Unauthorized use of the database or application.
- Accidental modifications and deletions.
- Confidentiality and integrity breeches for data in data transport and physical storage.
- Disasters.

Many new Internet applications are being developed as part of e-commerce initiatives. It is imperative to ensure that any access to data is performed by authorized sources. Implementing and monitoring security policies are crucial for each project and across the infrastructure.

There are various security models that can be deployed when implementing an Internet application. The appropriate model to deploy can be determined primarily based on the security requirements of the data being accessed (see Figure 21).

- A low security model only publishes basic information to the Internet. Typically no database is present in this model.
- A medium security model typically consists of data access methods and dynamic web pages. Application services are located on the internal network and only authenticated processes will be allowed to access databases behind the firewall. In this model, a replicated database may be located in the demilitarized zone (DMZ), but the authoritative source must be behind a firewall. Impact from security incidents on replicated data is minimized if the database in the DMZ is periodically refreshed from the authoritative source.
- A high security model is typically implemented when confidential, private, or protected data is accessed. This model provides access to restricted databases behind firewall by authenticated processes only. Data deemed confidential or private might need to be encrypted during transmission. In this situation, a Secured Socket Layers (SSL) connection or a Virtual Private Network (VPN) using technologies such as IPSec is recommended. The database must not be replicated in the DMZ.

For more information about security, refer to the Security and Directory Services Architecture chapter.

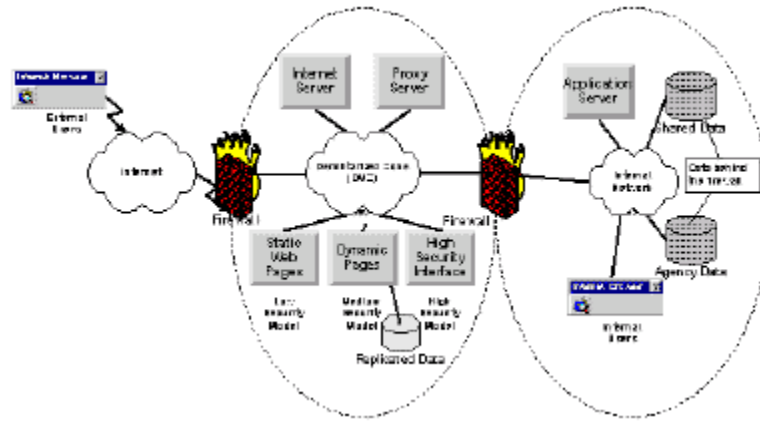


Figure 21. A sample security model including firewalls.

Security services are key to data security. As discussed in the Security Architecture, security services comprise the following processes:

- Identification. The process of distinguishing one user from all others.
- Authentication. The process of verifying the identity of the user.
- Authorization and access control. The means of establishing and enforcing rights and privileges allowed to users.
- Administration. The functions required to establish, manage, and maintain security.
- Audit. The process of reviewing system activities that enables the reconstruction and examination of events to determine if proper procedures have been followed.

For more information about security, refer to the Security Services chapter.

In many applications today, authentication is handled by the application when the user first connects. This practice is used to minimize user account administration, so a user account is defined centrally, and accounts and passwords are not maintained in multiple locations. In the back end, when databases are accessed, a generic user account is used instead of a specific user account for authorization purposes. (See Figure 22.) When this method is deployed, it is important to implement an audit process within the application to ensure that the activities of each user are captured.

When a generic account is used, the individual account is not automatically communicated to the database and tracked in the transaction log. Without an audit process in place, when a record or group of records is updated or deleted, it may not be possible to know which user performed the modifications. When a true threat is realized, it is important to be able to recover the lost data and track the user account that was used inappropriately. A best practice for the application is to capture key information about each user who modifies or deletes records. This information can be stored in the database, capturing old and new information, or the deleted record, along with transaction activity for an update or delete of a record. An audit process may not be necessary during a read-only transaction, as the data is not being modified, unless sensitive data is involved.

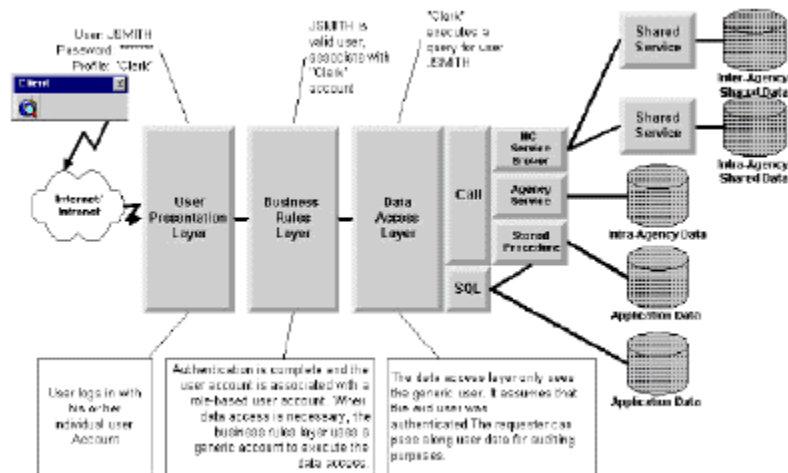


Figure 22. A generic user example

Implementing backup and recovery procedures is also a crucial process for data security. A backup can limit the loss of data that may occur due to disaster, theft, intrusion, or accident. Data stored on laptops must also have a backup and recovery plan and key information must not be stored on a laptop without encryption or password protection.

Protecting a database server is also a consideration when implementing and supporting a database. It is possible to protect a database from unauthorized access. Tools such as security access control and intrusion detection software can scan servers for potential weaknesses and detect inappropriate use as it occurs such as multiple invalid login attempts.

A firewall can also be used to protect a database server. A firewall can limit the access to a server by restricting the addresses and/or requests to the database server. For example, only certain programs can be allowed access to the server, restricting individual ad hoc usage. Access can be limited to specific application servers or processes as well.